Big Data Algorithms and Analysis Midterm Exam

Name: Yuding Liang No.: 116033910017

1. (Designer: Juntao Wang. You are required to answer this question)

(a) Assuming t is the time that CPU to access a profile in memory.

For the quick selection, we assume that n is the total number of profiles and k is the number of groups that we want to split into, $S(k)$ is the number of comparison required to sork k numbers, $C(n)$ is the number of comparison required by the basic algorithm. So the total number of comparison is $C(n) \leq \left\lfloor \frac{n}{k} \right\rfloor S(k) + C\left(\left\lfloor \frac{n}{k} \right\rfloor\right) + n - 1 + C(\frac{3n+k}{4})$.

For $k \geq 5$ and $n > (k+2)/(k+4)$: $C(n) \leq \left(1 + \frac{S(k)}{k}\right)\left(\frac{4k}{k-4-\frac{k+2}{n}}\right)n$, and the optimal value of k is 21, so $C(n) \leq \frac{348n}{17} < 20.5n$, so the Running time of quick selection in this problem is $O(1000 \times 20.5 \times nt) = O(20.5 \times 2^{56}t)$

The worst case of one-pass algorithm is that every time we read a new number it is the median of data that is read before, so we need $\log k$ times comparison when a new number comes, k is the number of files stored in memory. So the time complexity of the worst case in one-pass algorithm is $O(n \log k)$, where n is the size of data and k is the number of data that can be stored in memory. Running time is $O(10000 \times 2^{48}t) = O(2^{62}t)$

(b)

I find an existing algorithm(An Efficient Algorithm for the Approximate Median Selection Problem) whose running time is linear in the length n of the input, and it performs fewer than $\frac{4}{3}n$ comparisons and $\frac{1}{3}n$ exchanges on the average.

TA(a): And pseudo-code of this algorithm is as follows:

For quick selection: The running time is around 10*c1*2^56t = 20*c1*2^55t.
c1 is the constant of the times visiting each element in order to divide them into two parts.

For one-pass algorithm: In order to have the accuracy guarantee, we need to keep track on sqrt(2^46) profiles, i.e., 2^23 profiles with totally 2^33B space. While the memory space is 2^32B, only half of the profiles can be stored in memory and the rest half has to be stored in SSD. When a new number comes, the worst time to insert thenumber and to maintain the data in track is c2*logk. Therefore, a more precise estimation of the running time is (n*c2*logk*(1000+1)/2*t), which is around 2^46*c2*23*500t = 23*c2*2^55t.

Therefore, your decision should depend your analysis of the constant estimation.

(b)

It's open. One approach is to randomly sample a set of numbers and find the sample median as the true median. Although it has been proved that if the sample size approaches to infinite, the sample median will follow the normal distribution. The converge rate is still unknown to me.

```
SELECTION_SORT ($A, Left, Size, Step$)
    This procedure sorts Size elements of the array $A$ located at positions $Left, Left+Step, Left+2\cdot Step \ldots$.
    for ($i = Left$ ; $i < Left + (Size-1)\cdot Step$; $i = i + Step$)
            $Min = i$;
            for ($j = i + Step$; $j < Left + Size \cdot Step$; $j = j + Step$)
                    if ($A[j].Key < A[min].Key$) then $min = j$;
            end for;
            SWAP ($A[i], A[min]$);
    end for;

APPROXIMATE_MEDIAN_ANYN ($A, Size$)
    This procedure returns the approximate median of the array $A[0, \ldots, Size-1]$.
    $LeftToRight = False$;   $Left = 0$;   $Step = 1$;
    while ($Size > Threshold$) do
        $LeftToRight$ = Not ($LeftToRight$);
        $Rem = (Size \bmod 3)$;
        if ($LeftToRight$) then $i = Left$;
                        else $i = Left + (3 + Rem)\cdot Step$;
        repeat ($Size/3 - 1$) times
            Triplet_Adjust ($A, i, Step$);
            $i = i + 3 \cdot Step$;
        end repeat;
        if ($LeftToRight$) then $Left = Left + Step$;
                        else $i = Left$;
                            $Left = Left + (1 + Rem)\cdot Step$;
        SELECTION_SORT ($A, i, 3 + Rem, Step$);
        if ($Rem = 2$) then
                if ($LeftToRight$) then SWAP ($A[i+Step], A[i+2\cdot Step]$)
                                  else SWAP ($A[i+2\cdot Step], A[i+3\cdot Step]$);
        $Step = 3 \cdot Step$; $Size = Size/3$;
    end while;
    SELECTION_SORT ($A, Left, Size, Step$);
    return $A[Left + Step \cdot \lfloor (Size-1)/2 \rfloor]$;
```

And the probability is:

$$P_a^{(r)} = \left(\frac{2}{3}\right)^r \frac{3^{a-1}}{\binom{n-1}{a-1}} \sum_{b_r, b_{r-1}, \cdots, b_3} \prod_{j=2}^{r} \sum_{i_j \geq 0} \binom{b_j - 1}{i_j} \binom{3^{j-1} - b_j}{b_{j+1} - 2b_j - i_j} \frac{1}{9^{i_j}}$$

2. (Designer: Xiaofu Huang and Yun Guo)

(a) whether the quick hull algorithm is output sensitive algorithm or not? Explain why. (An algorithm is output sensitive if its running time depends on the size of the output.)

Solution:

The quick hull algorithm is output sensitive.

Let r be the number of recursive input, N be the total number of input, so $r \leq N$. Running time of one recursive operation is $O(r)$. Because we can find a vertex of convex hull during every recursion and we have h recursions totally, the time complexity is $O(Nh)$. The time complexity is related the size of output, so quick hull algorithm is output sensitive.

(b) If h is the number of the points on the convex hall. For points independently drawn from a circular Gaussian distribution, it is known that E(h) = O(log(n)) for sufficiently large n, where n is the number of all points. What's the time complexity of the quick hall algorithm in this case?

Solution:

From question (a), we know the time complexity of quick hull is $O(Nh)$, so the time complexity is $O(NE(h)) = O(NlogN)$ in this case.

3. (Designer: Xiaofu Huang and Yun Guo)

(a) Briefly describe the Kirkpatrick and Seidel algorithm for finding convex hall.

Solution:

<1> Find the median of x-coordinates of all points x^*

<2> Find the upper/bottom tangent $t(\overrightarrow{U_l}, \overrightarrow{U_r})$/ $t(\overrightarrow{b_l}, \overrightarrow{b_r})$ of left points and right points

<3> Find upper and bottom tangents the points left to U_l/b_l recursively

<4> Find upper and bottom tangents the points right to U_r/b_l recursively

(b) Given n points in the plane and a partition principle below. Prove that the Kirkpatrick and Seidel algorithms complexity is $O\left(min_{S_i} \sum_{i=1}^{k} |S_i| log\left(\frac{n}{|S_i|}\right)\right)$.

Solution:

Because we split the set into many groups, so we can compute the bound of every group first can sum over all of them finally.

Considering a set S_i. The set's contribution to the work done at recursion level j is linear in the number of points in S_i that haven't deleted by the algorithm. So there are at most $|S_i|$ point left. And according to the claim, the bound of $|S_i|$ is better than the bound of $\frac{n}{2^j}$ for the first $\approx \log_2 \frac{n}{|S_i|}$ recursion levels. So the running time of the KS algorithm is $\leq |S_i| \log \frac{n}{|S_i|} + \frac{|S_i|}{2} + \frac{|S_i|}{4} + \cdots$ that is $O\left(|S_i| \log log \frac{n}{|S_i|}\right)$.

Finally sum over all the subsets, we get KS algorithms complexity is $O\left(min_{S_i} \sum_{i=1}^{k} |S_i| log\left(\frac{n}{|S_i|}\right)\right)$.

4. (Designer: Wenbo Li and Qipeng Zhang)

(a) Prove that landmarks-cover problem is NP-hard. (Hint: reduce the vertex cover problem to this problem.)

Solution:

<1> L is a solution for LANDMARK-COVER. Consider the set of all 1-hop neighbors and each pair is connected by a unique shortest path of length 1. Because the shortest paths of every pair is

exactly the edges of the graph. So the solution L covered all edges and L is also the solution of VERTEX-COVER.

<2> V' is a solution for VERTEX-COVER. $(s,t)\epsilon V\times V$ is a pair of vertices and $p_{s,t}$ is any shortest path between s and t. According to definition, some vertices of V' should be on the edges of $p_{s,t}$, so that V' is also a solution to LANDMARK-COVER.

(b) Design a (heuristic) method to generate the landmarks set, and analyze the potential fault.

Solution:

Assuming that we want to select d vertices to be the landmarks.

<1> We firstly sort the nodes of the graph by decreasing degree

<2> Then we select landmarks L iteratively according to the rank that created in step 1.

<3> For each landmark L, if the distance between L and other selected landmarks is equal to or less than h(here h is the parameter decided by user), we will discard L and return to step 2. Otherwise, we add L into the set of landmarks.

<4> We repeat the step 2 and step 3 until we find d landmarks.

5. (Designer: Jinli Zhong and Tong Yin)

(a) come up with some ingenious algorithm to reduce the times of comparisons in average without harming the correctness. (Hint: you can precompute and save the distances of each pair of points in the data set, and when a query comes, you need not to compute these distances if you need to use them.)

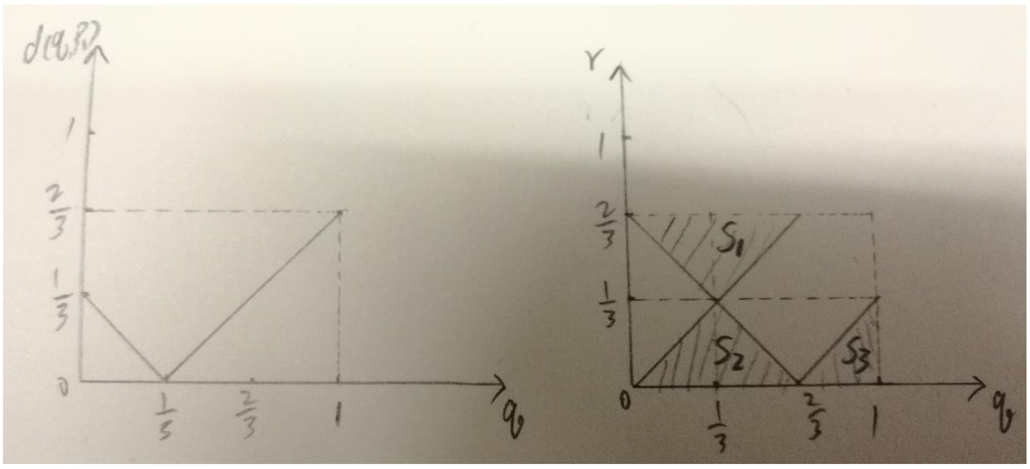
Solution:

We can precompute the distance of all pairs of points set, then use triangle inequality to reduce time. For example, when $\mathrm{d}(q,p_1)+d(p_1,p_2)<r$, here must be $\mathrm{d}(q,p_2)<r$ since the third edge's length must be smaller than the sum of the other two edges' length. And if $|\mathrm{d}(q,p_1)+d(p_1,p_2)|>r$, $\mathrm{d}(q,p_2)$ must be larger than r and we can exclude the point d from result.

(b) calculate in expectation how many comparisons are reduced by your algorithm if we consider the matrix space [0, 1], a set of two points p1 = 1/3 , p2 = 2/3 and that q is uniformly drawn from [0, 1] and r is uniformly drawn from [0, 2 3].

Solution:

We can first compute the distance of q and p1, and this distance is shown in the left figure. There are two situations that we don't need to compute the distance of q and p_2. The first one is $\mathrm{d}(q,p_1)+d(q,p_2)<r$ and the other one is $|\mathrm{d}(\mathrm{q},\mathrm{p}_1)-\mathrm{d}(\mathrm{p}_1,\mathrm{p}_2)|>\mathrm{r}$. In first situation, we can add p2 into result set directly and in the other one we have to exclude p2.Assuming that the total times of comparison is 1 without these tricks. Then the probability of avoiding computing the distance of q and p2 can be calculated from the right figure. The S1 is corresponding to the first situation, S2 and S3 are the other situation. According to the right figure, we can get that the probability is $\frac{1/9+1/9+1/18}{2/3}=\frac{5}{12}$.



6. (Designer: Atta Ul Munim Zaki)

It is impossible to cover all 269 million users in one survey due to the time and money. So we have to select some of them. There are three definitions that can help us to sample from the whole data. They are estimated response rate, margin of error and confidence level. The estimated response rate is the percent of those asked to take part in the survey. The margin of error is a statistics expressing the amount of random sampling errro in a survey's results. At last, the confidence level refers to the percentage of all possible samples that can be expected to include the true population parameter. We choose 20%, 2% and 95% are the value of above three definitions, respectively. With the help of website CheckMarket Calculator online, the required size of sample is 2401, so we need 2401/20%=12005 people to finish the survey. We can know the sample size n and the margin of error according to $\mathrm{e}=z^{*}\times\frac{\sigma}{\sqrt{n}}$ after the survey. And we can find the value of z^{*} from the following table:

Confidence level (%)	z^{*}
80	1.28
90	1.645
95	1.96
98	2.33
99	2.58

σ is the population standard deviation.

7. (Designer: Wei Yi and Huang Bo)

(a) Prove ∃c ∈ N, H(s) ≤ |s| + c for any finite string s.

Solution:

Define M:="On input y, output y" So $|(M, s)| \leq |M| + |s|$. And $\mathrm{H}(\mathrm{s}) \leq |M| + |s|$. We can find a number c ∈ N that $\mathrm{c} = |M|$. Proof finish.

(b) Prove ∃c ∈ N, H(< x, y >) ≤ |x| + |y| + 2 log(|x|) + c for any a pair of finite strings x and y.

Solution:

Let $M_x(x') = x \; where \; |(M_x, x')| = H(x), \; M_y(y') = y \; where \; |(M_y, y')| = H(y)$

Consider M that frist runs $M_x(x')$ then $M_y(y')$.

<1> $|(x, y)| = 2\,floor(log|x| + 1) + |x| + |y| + 2 \leq 2log|x| + |x| + |y| + 4$

<2>

$$\left|\left(M, \left((M_x, x'), (M_y, y')\right)\right)\right| \leq 2|M| + \left|\left((M_x, x'), (M_y, y')\right)\right|$$

$$\leq 2|M| + 2\log\left(H(x)\right) + H(x) + H(y) + 4 < 2\log(|x|) + |x| + |y| + 2|M|$$

We can find a number c that satisfies $\mathrm{c} = 2|M|$.

8. (Designer: Haixing Huang and Zhen Meng)

Solution:

Algorithm:

We firstly select all vertices whose weight are larger than k, and delete all edges related to it. We also need to check the vertex that is on the other side, if its weight is larger than k too, there won't be any satisfied vertex cover. If its weight is smaller than k, we add this vertex in to the set of vertex cover and delete the corresponding edge.

Then we should reduce the weighted vertex cover to an unweighted vertex cover. For each vertex $\mathrm{v} \in \mathrm{V}$, suppose the weight $\mathrm{w}(v) = u$, we replace the v with a cluster of u vertices and don't add any edges among these u vertices. If there is an edge between the vertex v and t, and the v', t' are the clusters of these two vertices respectively, we connect every vertex of v' to every vertex of t'.

Now, we can use the kernelization to generate vertex cover of new graph. And in new graph, either all vertices of a cluster are in a minimum vertex cover or none of them is. So the vertex cover of new graph can be converted to the vertex cover of original graph.

Time complexity:

Let $\mathrm{t}(k, n)$ be the time to solve unweighted vertex cover problem, so the running time of this algorithm is $\mathrm{t}(k, wn) = O(t(k, kn))$.

9. (Designer: Yuding Liang and Ze Chen)

(a)

If $0 \leq X \leq 1/18$: $Y_{1,3}(X) = Y_2(X)$, $Y_{2,3}(X) = Y_3(X)$ and $Y_{3,3}(X) = Y_1(X)$
If $1/18 \leq X \leq 1/5$: $Y(X) = Y_3(X)$, $Y_{2,3}(X) = Y_2(X)$ and $Y_{3,3}(X) = Y_1(X)$
If $1/5 \leq X \leq 5/12$: $Y_{1,3}(X) = Y_3(X)$, $Y(X) = Y_1(X)$ and $Y(X) = Y_2(X)$
If $5/12 \leq X \leq 1$: $Y_{1,3}(X) = Y_1(X)$, $Y_{2,3}(X) = Y_3(X)$ and $Y_{3,3}(X) = Y_2(X)$

(b)

i. $F_{r,n}(x) = \Pr[X_{(r)} < x] = Pr[X_{(r+1)} < x] + Pr[X_{(r)} < x, X_{(r+1)} \geq x]$
$$= F_{r+1,n}(x) + C_r^n(F(x))^r[1 - F(x)]^{n-r}$$
ii. $F_{r,n} = Pr[X_{r,n-1} < x] + Pr[X_{r-1,n-1} < x, X_{r,n-1} \geq x] \times F(x)$
$$= F_{r,n-1}(x) + C_{r-1}^{n-1}(F(x))^r[1 - F(x)]^{n-1-(r-1)} \times F(x)$$
$$= F_{r,n-1}(\mathrm{x}) + C_{r-1}^{n-1}(F(x))^r[1 - F(x)]^{n-r}$$

(c)

By observation, according to $Y \leq y$ we know in n observations at least r of them are smaller than x, and according to $Z > z$ we know in n+1 observations at least (n+1-r) bigger than y, so there are exactly no observation between (y, z), so by definition of joint distribution:

$\Pr\{Y \leq y, Z > z\}$
$$= \sum_{j=1}^{r+1} \sum_{i=r}^{j-1} \Pr\{exactly\ i\ X_i \leq y, exactly\ j\ X_j > z\}$$
$$= Pr\{exactly\ X_{r,n} \leq y, exactly\ X_{r+1,n+1} > z\} = C_r^n(F(x))^r[1 - F(y)]^{n-r+1}$$

10. (Designer: Renjie Jin and Jie Li Liang)

(a) Calculate the distribution of $\mathrm{X}_{1,10}$.

Solution:

$$\Pr(\mathrm{X}_{1,10} \geq \mathrm{x}) = (1 - F(x))^{10} = (1 - (1 - e^{-\lambda x}))^{10} = e^{-10\lambda x}$$

(b) Suppose that $\mathrm{X}_{2,10} = 3$ after each variable is revealed, what is the distribution of $\mathrm{X}_{3,10}$ now.

Solution:

$x_{3,10} = 3 + (x_{3,10} - x_{2,10})$, $(x_{3,10} - x_{2,10}) \sim \exp(10 - 2) \sim \exp(8)$,

So $f_{3,10}(x) = 8e^{-8(x-3)}, x > 3$

(c) Assuming that the time to generate a variable from the exponent distribution is O(1), design an algorithm to generate the k-th smallest number in time complexity O(k).

Solution:

We should to generate k numbers from the exponent distribution $y_1, y_2 \ldots, y_k$, then we generate k-th smallest number by $x_{i+1} = x_i + y_i (x \geq 0)$, and x_i follows exponent distribution.

Assume that $x_0 = 0$ and let $y_i = x_i - x_0$;

$\mathrm{f}(y_i) = (n - i + 1)e^{-(n-i+1)(y_i)}$, $x_{n-i+1} = y_i + y_{i+1} + \cdots + y_n$, and because the time of generating a y_i is $\mathrm{O}(1)$ and we need to generate y_i k times to generate k-th smallest number, the time complexity is $\mathrm{O}(\mathrm{k})$.